The Thesis committee for Chia-Wen Cheng
certifies that this is the approved version of the following Thesis:

Artistic and Semantic Progressive Image Coding

APPROVED BY
SUPERVISING COMMITTEE:

Philipp Krähenbühl, Supervisor

Kristen Grauman

Artistic and Semantic Progressive Image Coding

by

Chia-Wen Cheng

Thesis

Presented to the Faculty of the Graduate School

of The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

May 2019

Acknowledgments

First, I would like to thank my advisor Philipp Krähenbühl for sharing his knowledge, constantly trying to improve my work, and showing me how to do rigorous research. Although I got stuck several times, he patiently helped me explore various directions. Without his continuous guidance, this thesis would not have been possible. I am also grateful to my thesis committee member, Kristen Grauman for her detailed and insightful feedback that strengthened this thesis.

I would like to thank my labmate Chao-Yuan Wu for encouraging me to do research, extensively discussing with me, and offering help in my many previous experiments. He served as a mentor throughout my research. From him, I also learned how to deliver a good presentation and improve writing skills.

Special thanks to Kuan-Lun Tseng for helping me build the web interface for user studies. My thesis will not be complete without this component.

I am grateful to all the participants who volunteered their time to participate in my user studies.

Finally, my peers in the CS department deserve great thanks. Thanks to Shih-Yun, Wei-Lin, Chia-Chen, Shailee, Wei-Ju, and Yu-Chuan for giving me advice on research, which helped me persevere through difficulties and finish this thesis.

Artistic and Semantic Progressive Image Coding

Chia-Wen Cheng, MSCompSci

The University of Texas at Austin, 2019

Supervisor: Philipp Krähenbühl

Progressive image coding provides web users a faster full-image preview when only a small fraction of the file has been transmitted. JPEG is the most widely used progressive compression technique. However, the compressed quality is low at low bitrates and it entirely ignores the semantics of images. We propose to render different artistic visual effects and leverage semantic information in the images. The key insight is human visual system is more sensitive to luminance and semantic salient objects. Our learning based progressive coding approach learns to encode a smooth transition from grayscale images to color images. During decoding, our approach will display sharp grayscale images first instead of blurry color images, which allows viewers to preview images faster. In addition, our approach can allocate more bitrates to important objects according to segmentation masks. The approach can be extended to encode other artistic styles such as mosaic style and users can easily create their own decoding patterns. Our progressive image coding method generates clearer content than JPEG and previous learning based progressive compression method in both quantitative metrics and user studies.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

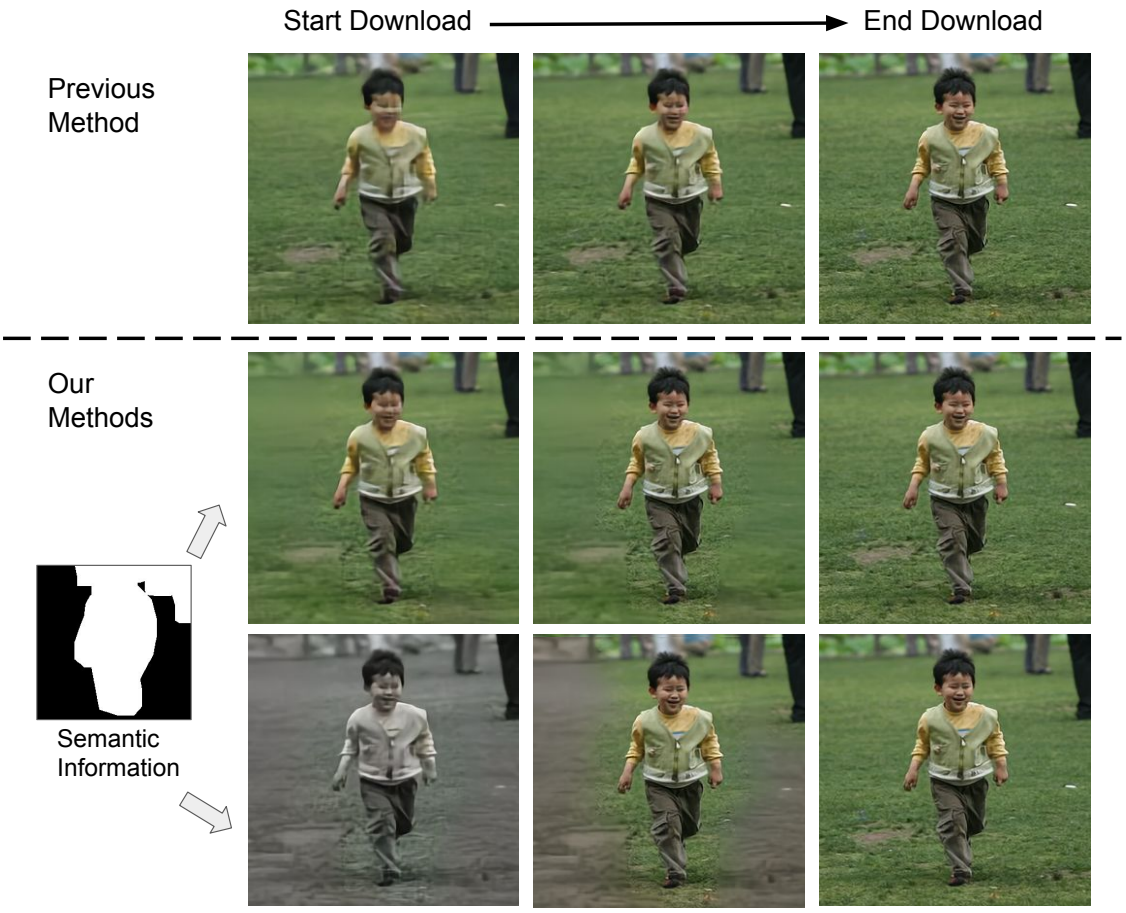


Figure 1.1: Progressive coding improves user experience in viewing images on the web. Our proposed methods provide artistic loading effects and semantic variant loading.

Images take up massive amounts of internet bandwidth. We share moments, deliver ideas, and create lively conversations on social network platforms with the aid of images. To facilitate image transmission over the internet, we need strong image compression techniques to optimize images' file size and visual quality. JPEG is the

most dominant image compression method used on the internet. Its standardization dates back to as far as 1992. Although it has certainly served us well, the web is a hundred times more saturated than it was in the '90s [14]. It will be valuable to re-examine this old technique and design a newer compression method to adopt to current crowded internet traffics.

A large fraction of web platforms deploy progressive JPEG to improve user's experience when viewing images on the internet. For example, Twitter [3] and Yelp [5] ship progressive JPEG on their platform, and Facebook ships progressive JPEG for their iOS app [7]. Non-progressive coding methods sequentially decode an image from top to bottom, so users have to wait until the full file is decoded, which hurts the user experience on the web. This becomes more problematic when users are on slow connections. On the other hand, progressive coding methods instantly display a rough reconstruction of the whole image and slowly refine the image. Users can see what's in the image even when only a fraction of the file has been transferred, and decide whether they want to wait for it to fully load or not. Progressive coding benefits applications used to view images from the internet or from data sources with limited bandwidth. It is especially useful for large image files.

Traditional method, JPEG, performs poorly at low bitrates. It imposes blocky, ringing and color bleeding artifacts on original images, which compromises our viewing experience. This problem also exists during progressive decoding. Besides, JPEG decodes all parts of the image in the same speed. However, users may only need to see some important regions and then they can decide whether to wait for the image to fully load or not. The latest learning based progressive method [21] offers clearer previews than progressive JPEG using convolutional recurrent networks. However, it only provides one progressive reconstruction style: blurry to sharp (see the top row in

Figure 1.2) by simply optimizing L1 loss between reconstruction and original image. It does not investigate other possible progressive reconstruction styles or other losses. It also decodes all regions in an image in the same speed. In this thesis, we explore more flexible progressive coding methods to improve user experience. Inspired by the recent success of learning based image compression methods [6, 10, 17, 21], we extend the state-of-the-art learning based progressive method [21] to more progressive dimensions: color, resolution, and spatial. In [21], they employ an autoencoder to encode an image multiple times. Each time, the autoencoder encodes the residual between current reconstructed image and original image. The autoencoders across time steps form a recurrent framework with convolutional recurrent networks in both encoder and decoder. They optimize L1 loss on the residuals, which displays a transition from blurry to sharp during progressive decoding. We modify their training objective for more flexible progressive coding, creating artistic visual effects during decoding such as grayscale to color style and mosaic style (see the middle and bottom rows in Figure 1.2). We create progressive targets for one image and guide the recurrent framework to match different interpolation of progressive targets in different time steps. The whole recurrent framework is an autoencoder, but at a single time step it is not an autoencoder anymore. Further, we incorporate semantic information to the image for spatially variant progressive decoding. We call it semantic progressive decoding. We mask out partial code during decoding to decode different regions in a different order. An image can be progressively decoded either in full image or in parts without changing original encoding procedure. More interestingly, users can interact with the codec. They can specify semantic salient regions in images to create their own reconstruction results (see Figure 1.3, Figure 1.4, and Figure 1.5). This spatially progressive decoding method may be used in video streaming or video chats, so when

users are on the low connections, they can specify where they want to see more clearly instead of seeing a blurry scene.

We compare our algorithm to previous state-of-the-art learning based progressive method [21] and JPEG. We evaluate all algorithms on a standard dataset of uncompressed images: Kodak dataset [1] and on a large dataset with segmentation masks: MS COCO dataset [11]. In addition, we conduct user studies to understand how human subjects really perceive reconstructed images, and how well human visual perception correlates with existing image distortion metrics. Both of the quantitative and qualitative results show that our grayscale to color encoding yields the clearest reconstruction at the low bitrate regime, and semantic progressive decoding offers a more perceptually pleasing preview when only a small percentage of file is decoded.

Figure 1.2: Different progressive coding methods. Top: Progressive reconstruction results by [21]. It displays images from blurry to sharp . Middle: Our progressive reconstruction results in color dimension. It shows a transition from grayscale to color. Bottom: Our progressive reconstruction results in resolution dimension. It renders images from low resolution to high resolution. We call it mosaic style because images look like they are assembled by small pieces of colored tiles.

Figure 1.3: Semantic progressive decoding results with the encoding method of [21]. The vanilla decoding result is in the top row of Figure 1.2. We decode the same code in two different orders. The leftmost column shows user specified semantic salient regions (white regions) in the image. The first one specifies face, and the second one specifies bananas. During decoding, salient regions are reconstructed faster than non-salient regions.

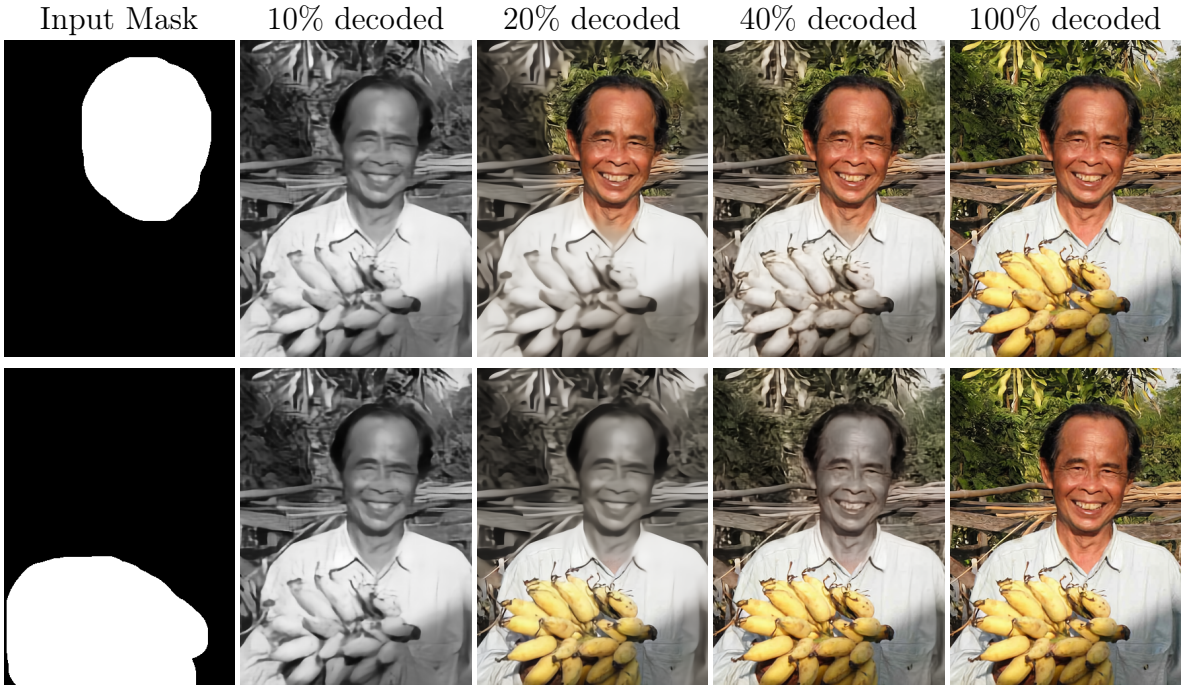


Figure 1.4: Semantic progressive decoding results with grayscale to color encoding. The vanilla decoding result is in the middle row of Figure 1.2. We decode the same code in two different orders. The leftmost column shows user specified semantic salient regions (white regions) in the image. The first one specifies face, and the second one specifies bananas. During decoding, salient regions are reconstructed faster than non-salient regions.

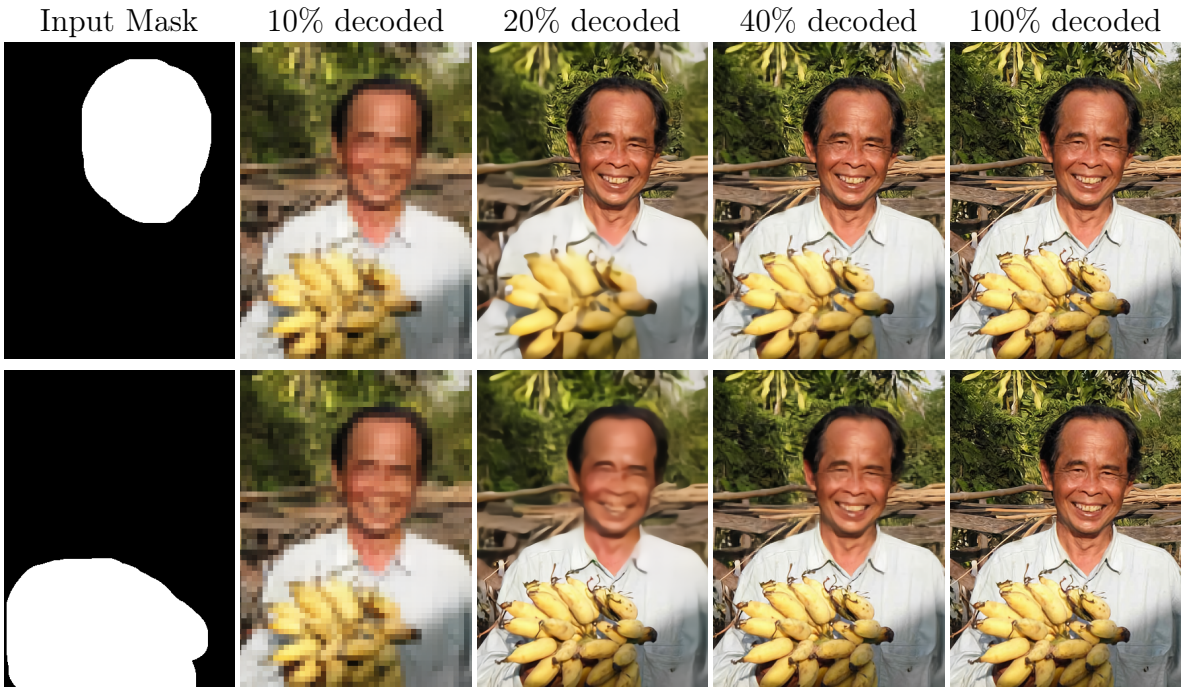


Figure 1.5: Semantic progressive decoding results with mosaic style encoding. The vanilla decoding result is in the bottom row of Figure 1.2. We decode the same code in two different orders. The leftmost column shows user specified semantic salient regions (white regions) in the image. The first one specifies face, and the second one specifies bananas. During decoding, salient regions are reconstructed faster than non-salient regions.

Chapter 2

Related Work

2.1 Traditional Image Codecs

Existing lossy image compression standards include JPEG [22], JPEG 2000 [19], and BPG [8]. JPEG is the most commonly used method for storing digital images. JPEG first transforms an image from RGB to Y'CbCr color space, disentangling luma and chroma signals. Then, it divides the image into 8×8 blocks, and applies discrete consine transformation (DCT) to each block, converting the image from the spatial domain to the frequency domain. Finally, JPEG quantizes the DCT coefficients. The quantized DCT coefficients are further compressed with Huffman encoding.

JPEG 2000 was created with the intention of replacing JPEG. JPEG 2000 decomposes an image into non-overlapping blocks which can be of arbitrary size. A newly-designed, wavelet transformation is applied to each patch to create pyramid representation. The quantized code blocks are then encoded with a context-driven binary arithmetic coder. Although JPEG 2000 achieves a large improvement at low bit-rate compression, this format has not been widely adopted yet.

BPG is a new image format based on HEVC, the state-of-the-art video compression standard, and it outperforms JPEG and JPEG 2000. While JPEG and JPEG 2000 encode each block in the image independently, BPG utilizes redundancy between blocks and encodes only the differences between blocks.

To enable faster preview of images on the web, many image codecs support progressive coding. Progressive JPEG is the most commonly used format on the web.

A normal JPEG encodes an image block by block so the image will be loaded from the top to bottom line by line. On slow internet connections, top of the image will be revealed first and a large fraction is left whitespace. In contrast, a progressive JPEG shows the entire image right away, from low resolution to high resolution. This is done by progressive encoding, encoding low frequency signals of all blocks first, then high frequency signals. While our method also address progressive coding, we provide more dimensions of progression: color, resolution and spatial location, which creates artistic effects.

2.2 Deep Image Compression

Recent work shows deep neural network based methods outperform traditional hand-designed algorithms in image compression [6, 10, 13, 17, 20, 21]. One common approach is to train a convolutional autoencoder with a binary bottleneck layer to minimize the distortion between original and decompressed image [6, 10, 13, 17]. The disadvantage of this approach is that we need to train one autoencoder per compression rate.

Another approach is to train a convolutional recurrent neural network to progressively encode and decode images [20, 21]. This architecture supports variable compression rates without the need for retraining or for storing multiple encodings of the same image. Our work is built upon this type of network architecture but incorporates different loss objectives to create diverse visualization effects.

2.3 Content-Aware Image Compression

Contents of images have been exploited to achieve better compression ratios. Context-adaptive binary arithmetic coding (CABAC) [12], a form of entropy coding, exploits the content in binary code. It is used in the image compression standard, JPEG 2000, and in the latest video compression standards, H.264 and HEVC. It serves as a post-processing step to further compress binary codes by reducing inter-symbol redundancies. Operating on binary symbols, a probability model predicts the current symbol to encode according to already-coded symbols in the neighborhood called context. The probability model will be updated based on the actual coded value. If a symbol can be perfectly predicted by its context, we do not need to store it at all. Therefore, the more accurate the prediction is, the shorter the resulting encoded string is. CABAC has multiple probability models and they are selected adaptively based on the context. However, the selection mechanism and the probability models are hand designed and hand optimized.

Recent deep image compression work employs learned probability models to take larger context into account. Toderici *et al.* [21] train a PixelRNN [15] and Li *et al.* [10] train a 2D CNN as probability models to improve compression performance. Mentzer *et al.* [13] use a 3D CNN, and jointly train the probability model and the autoencoder to directly control the traid-off between the entropy of code and the distortion.

Another promising direction is to allocate different bit rates to different parts of the image. Li *et al.* [10] and Mentzer *et al.* [13] both learn an importance map with the autoencoder to guide bit rate allocation. More bits are allocated to the regions with strong edges or detailed textures while less to the smooth regions. Prakash *et al.* [16] incorporate semantic saliency map with JPEG. Agustsson *et al.* [4] propose

extreme image compression based on Generative Adversarial Networks (GANs) to synthesize unimportant regions given semantic label maps. However, when bit budget is constrained, the importance maps in [10] and [13] preserve details in unimportant regions such as leaves of tree or ripples in water but blur out important objects such as human face, which is not satisfying. We propose to directly use semantic labels to guide bit allocation on object level. The semantic labels can be predicted from a pretrained segmentation network or given by users. The Previous method that incorporates semantic labels [16] only tried with JPEG in normal compression setting. Different from their work, we incorporate semantic information in the latest learning based compression method and in progressive coding setting. GANs in [4] synthesize scenes totally different from the original ones, which violates the purpose of compression. However, our approach, at any bitrates, faithfully reconstructs images.

Chapter 3

Preliminary

Let $I \in \mathbb{R}^{H \times W \times 3}$ denote an image with height H and width W. Our goal is to progressively encode image I into a sequence of binary codes $b_t \in \{-1, 1\}^{N_t}$ for $t \in \{1, 2, \ldots, T\}$ and to decode codes to a sequence of reconstructed images $\hat{I}_t$ such that they are progressively reconstructed $d(\hat{I}_1, I) \leq d(\hat{I}_2, I) \leq \ldots \leq d(\hat{I}_T, I)$, where N_t is the number of bits produced at iteration t, T is the total iteration, and d is some distortion measure. We build on the model of Toderici *et al.* [21] to encode and decode an image progressively over T iterations.

The model consists of an encoder network E, a binarizer network B, and a decoding network D. E and D are stateful, containing Conv-LSTM layers, while B is stateless. The model architecture and parameters are shared across all iterations. Only the states in the encoder and decoder are propagated to the next iteration. At iteration t, E takes the residual r_{t-1} as input and produces an encoded representation. Then, B quantizes the encoded representation to binary representation $b_t \in \{-1, 1\}$. Finally, D takes the binary representation b_t and generates a residual reconstruction $D\left(b_t, h_{t-1}\right)$. The procedure can be written as follow:

$$
\begin{aligned}
r_0 &:= I, & \hat{I}_0 &:= 0 \\
b_t &:= B\left(E\left(r_{t-1}, g_{t-1}\right)\right), & \hat{I}_t &:= D\left(b_t, h_{t-1}\right) + \hat{I}_{t-1} \\
r_t &:= I - \hat{I}_t, & &\text{for } t = 1, 2, \ldots T
\end{aligned} \tag{3.1}
$$

where g_t and h_t are latent Conv-LSTM states at iteration t, $\hat{I}_t$ is the reconstructed

image, and r_t is the residual between I and the reconstruction $\hat{I}_t$. After T iterations, the network produces codes $\{b_1, b_2, \ldots, b_T\}$ and reconstructed images $\{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_T\}$. We use $L1$ loss to measure the distortion $d(\hat{I}, I) = \|\hat{I} - I\|_1$. The total training objective is to minimize the distortion at all the iterations $\sum_{t=1}^{T} \|I - \hat{I}_t\|_1 = \sum_{t=1}^{T} \|r_t\|_1$. Since the network repetitively encodes the residual between the constructed image and the original image, the resulting image sequence satisfies the constraint $\|\hat{I}_1 - I\|_1 \geq \|\hat{I}_2 - I\|_1 \geq \ldots \geq \|\hat{I}_T - I\|_1$.

Both the encoder and the decoder are fully convolutional networks. A $H \times W \times 3$ input image is reduced to an $H/16 \times W/16 \times L$ binary feature map in the bottleneck where L is the channel size. This results in each iteration representing $L/(16 \times 16)$ bit per pixel (bpp). The decoder upsamples the feature maps with the depth to space operation [18]. Toderici et al. [21] adopt a stochastic binarization during training to allow gradients backpropagate through the bottleneck. The binarization $b(x)$ of $x \in [-1, 1]$ is

$$b(x) = x + \epsilon \in \{-1, 1\}, \epsilon \sim \begin{cases} 1 - x & \text{with probability } \frac{1+x}{2}, \\ -x - 1 & \text{with probability } \frac{1-x}{2} \end{cases} \tag{3.2}$$

where ϵ corresponds to quantization noise. The gradient of $b(x)$ is 1 since $E[b(x)] = x$ for all $x \in [-1, 1]$. During testing, the stochastic binarization $b(x)$ is replaced by the fixed binarization $\tilde{b}(x)$

$$\tilde{b}(x) = \begin{cases} 1 & \text{if } x > 0, \\ -1 & \text{otherwise.} \end{cases} \tag{3.3}$$

Chapter 4

Approach

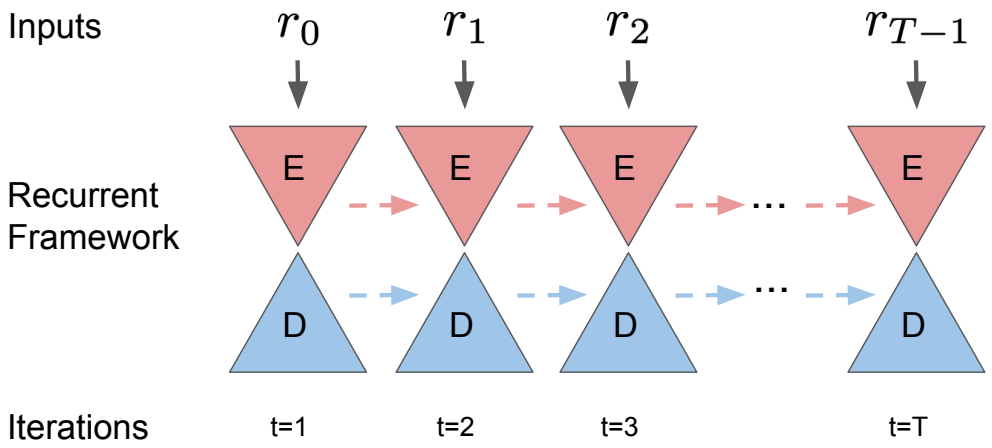


Figure 4.1: Our recurrent framework. The dashed lines denote the propagation of Conv-LSTM states.

We first discuss encoding different progressive styles to create various image rendering effects by modifying training objective, and then show semantic progressive decoding by spatial variant bit allocation, which aligns more consistently with human perception.

4.1 Progressive Style Encoding

Let I be an RGB image and T be the total number of progressive level, the number of times that the encoder and the decoder have to traverse. We choose $T = 10$ to achieve a reasonable reconstruction in all our experiments. We train a recurrent neural network based image compression model following Toderici *et al.* [21] to iteratively encode image I. Figure 4.1 shows the recurrent framework. We use

Conv-LSTM as our recurrent unit in the encoder E and the decoder D. The Conv-LSTMs are unrolled for $T = 10$ steps. At iteration t, the encoder E takes the residual between previous reconstructed image and the original image as input, $r_{t-1} = I - \hat{I}_{t-1}$, and the decoder D outputs the residual between previous reconstructed image $\hat{I}_{t-1}$ and current target image Y_t. In the original model, $Y_t = I$ across all iterations so D learns to produce $Y_t - \hat{I}_{t-1} = I - \hat{I}_{t-1} = r_{t-1}$, which is the same as the input to E. In short, the model is simply an autoencoder with input r_{t-1} and output $D(b_t, h_{t-1})$ at a time step, minimizing the loss $\|r_{t-1} - D(b_t, h_{t-1})\|_1 = \|I - \hat{I}_{t-1} - D(b_t, h_{t-1})\|_1 = \|I - \hat{I}_t\|_1 = \|r_t\|_1$.

Minimizing the total training objective $\sum_{t=1}^{T} \|r_t\|_1$ already gives us a sequence of progressive reconstruction $\{\hat{I}_1, \hat{I}_2, \ldots, \hat{I}_T\}$ from blurry to clear, a characteristic of L1 loss. How can we create other kinds of progressive style? For example, a continuous transition from grayscale to color, or mosaic style from low resolution to high resolution.

One way is to design a specific loss between the reconstruction and the original image for each style. Then we simply replace $L1$ loss in the training objective with the new loss. However, it is not trivial to hand design such loss. Instead, we find a simpler function F to decompose a fixed target I into N progressive targets

$$\begin{aligned} & Y^{(i)} = F_i(I), \text{ for } i = 1, 2, \ldots N \\ & \text{s.t. } \|Y^{(1)} - I\|_1 \geq \|Y^{(2)} - I\|_1 \geq \ldots \geq \|Y^{(N)} - I\|_1, \end{aligned} \tag{4.1}$$

The progressive targets will guide the reconstruction. Our new total training objective is

$$L := \sum_{t=1}^{T} \sum_{i=1}^{N} \lambda_t^{(i)} \|Y^{(i)} - \hat{I}_t\|_1, \tag{4.2}$$

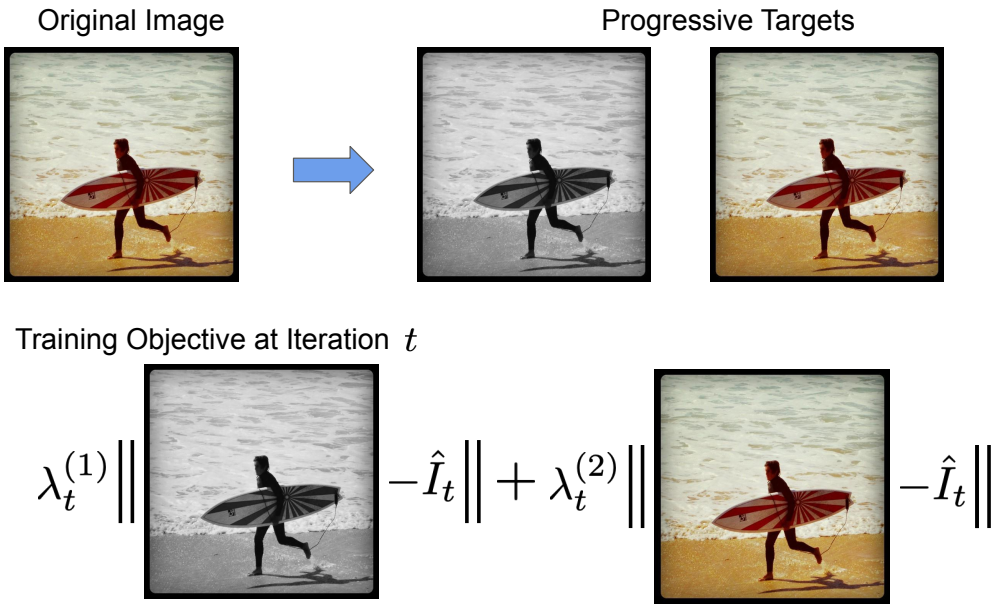


Figure 4.2: Training procedure for grayscale to color encoding.

where $\sum_{i=1}^{N} \lambda_t^{(i)} = 1$. A large $\lambda_t^{(i)}$ draws the reconstruction $\hat{I}_t$ toward $Y^{(i)}$. Note that our training procedure and inference remain the same as the formulation in 3.1. We only change the objective during training. The framework does not form an autoencoder at a single iteration anymore since the target is not the same as the input value. Instead, the whole framework is a single autoencoder.

We are interested in finding the rendering effects that can provide faster, clearer preview of an image, so we explore two kinds of progressive style: grayscale to color and mosaic style.

4.1.1 Grayscale to Color Encoding

Since human visual system is more sensitive to the black-and-white information than color [24], loading a grayscale image first may be beneficial. Figure 4.2 shows the training procedure for grayscale to color encoding. For an input image I, we

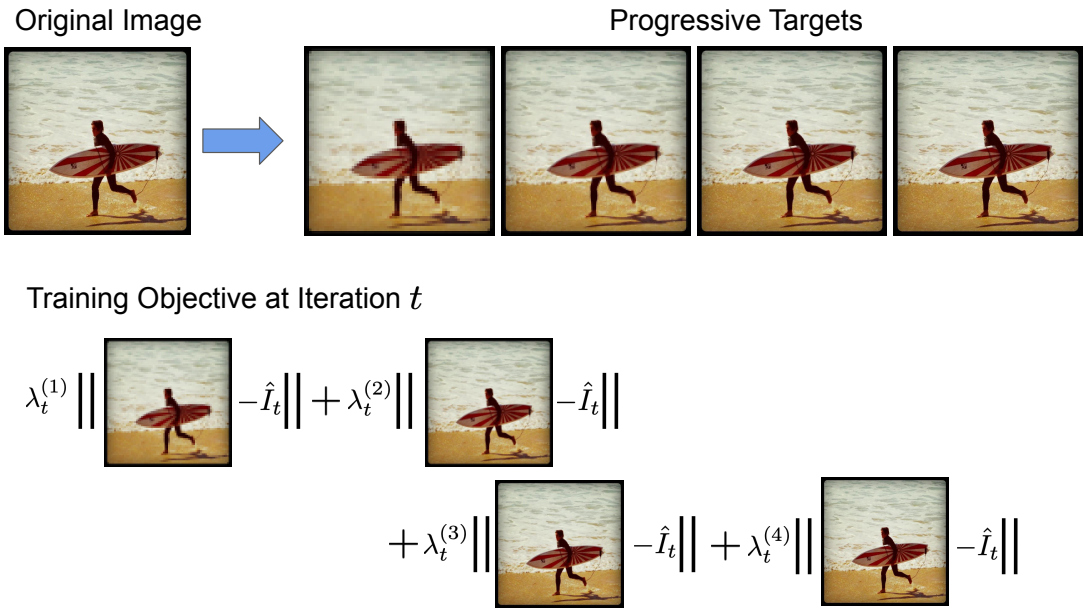


Figure 4.3: Training procedure for mosaic style encoding.

have two target images: the grayscale version I_G and the original one I. We convert the input image into Y'CbCr color space and use the luma component Y' as the grayscale version. We set $Y^{(1)} = I_G$ and $Y^{(2)} = I$. To generate a smooth transition from grayscale to color, we change the value of $\lambda_t^{(1)}, \lambda_t^{(2)}$ every two iterations, gradually decreasing $\lambda_t^{(1)}$ and increasing $\lambda_t^{(2)}$. We set $\lambda^{(1)}$ =(1.0, 0.6, 0.4, 0.2, 0) and $\lambda^{(2)}$ =(0, 0.4, 0.6, 0.8, 1.0).

4.1.2 Mosaic Style Encoding

A mosaic is a work of art made from the assembling of small pieces of colored glass, stone, or tile. We simulate this style by using a single color to represent a $c \times c$ block. Figure 4.3 shows the training procedure for mosaic style encoding. For an input image I, we apply $c \times c$ average pooling with a stride of c on it to extract the mean color of each block. This produces an output of resolution $\frac{H}{c} \times \frac{W}{c}$. Then we

use nearest neighbor interpolation to upsample it back to original resolution $H \times W$. The final image I_c is assembled by blocks of size $c \times c$, and each block is of the mean color of the original one in I. The larger c is, the more pixelated I_c looks. We set c = 8, 4, 2, 1 to create four target images $Y^{(1)} = I_8, Y^{(2)} = I_4, Y^{(3)} = I_2, Y^{(4)} = I_1$. To render image from low resolution to high resolution, we assign different values to $\lambda_t^{(1)}, \lambda_t^{(2)}, \lambda_t^{(3)}, \lambda_t^{(4)}$ in the first four iterations, and then remain the same for the following iterations. We set $\lambda_t^{(1)}$ =(1, 0, 0, 0), $\lambda_t^{(2)}$ =(0, 1, 0, 0), $\lambda_t^{(3)}$ =(0, 0, 1, 0), $\lambda_t^{(4)}$ =(0, 0, 0, 1) for $t = 1, \ldots, 4$, and $\lambda_t^{(i)} = \lambda_4^{(i)}$ for $t = 5, \ldots 10$.

4.2 Semantic Progressive Decoding

In the low bitrate regime, evenly allocating bits spatially will make an image all blurry. Instead, we should allocate more bits where human visual system is more sensitive to, e.g. foreground objects, and put less bits where human visual system is less sensitive to, e.g. background. For example, given an image with a person standing on a beach, it is more tolerable to blur out the sand, sea, and trees than human face. Even though there are a lot of low-level details such as edges and textures in the sand, sea, or trees, we would like to trade background details for clearer appearance of important objects. Similarly, during progressive decoding, we can first display clearer semantic salient objects with low-quality background and then refine the background.

Recall that at a single iteration the binary bottleneck b_t is a 3D feature map of size $H/16 \times W/16 \times L$. Let the receptive field of the bottleneck layer be of size $r \times r$. To fully decode an image, the decoder traverses the whole codes $\{b_1, \ldots, b_T\}$ one time. Over T iterations, we equally use $L \times T$ bits to represent each $r \times r$ receptive field. To use different bits to represent different regions, we decode important regions

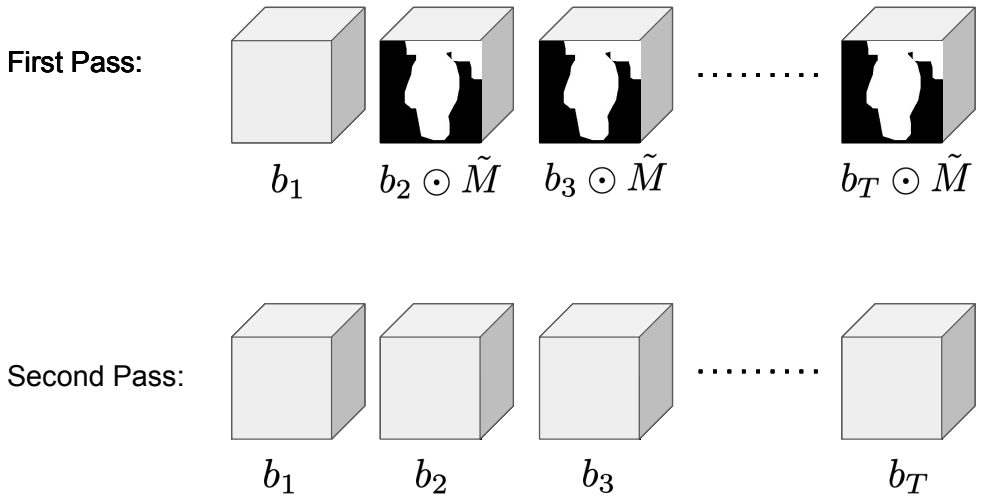


Figure 4.4: Semantic progressive decoding takes 2T iterations. In the first T iterations, the decoder decodes partial code (white regions) to reconstruct semantic salient regions. In the next T iterations, the decoder decodes the full codes to reconstruct remaining regions.

with more iterations and decode background with less iterations.

Given a binary mask $M \in \{0,1\}^{H/16 \times W/16 \times 1}$ indicating salient objects and background, we use it to mask the binary bottleneck to control bit allocation. Let $M_{i,j}$ denote the value of M at spatial location (i,j). $M_{i,j} = 1$ indicates salient objects, and $M_{i,j} = 0$ indicates background. We expand M into a mask $\tilde{M} \in \{0,1\}^{H/16 \times W/16 \times L}$ of the same dimensionality as b_t

$$\tilde{M}_{i,j,l} = M_{i,j}, \tag{4.3}$$

for $l = 0, 1, \ldots, L-1$.

The decoder traverses $\{b_1, \ldots, b_T\}$ two times to do spatially progressive decoding as shown in Figure 4.4. In the first pass, the decoder operates on masked codes

$\{b_1, b_2 \odot \tilde{M}, b_3 \odot \tilde{M} \ldots, b_T \odot \tilde{M}\}$

$$\begin{aligned} \hat{I}_1^{(1)} &= D\left(b_1, h_0^{(1)}\right), & \hat{I}_t^{(1)} &= D\left(b_t \odot \tilde{M}, h_{t-1}^{(1)}\right) + \hat{I}_{t-1}^{(1)}, \\ O_t &= \hat{I}_t, & &\text{for } t = 1, 2, \ldots T \end{aligned} \tag{4.4}$$

where $\odot$ denotes elementwise multiplication operation, and O_t denotes the final reconstruction at time step t. In this pass, we mainly reconstruct important objects. In the second pass, the decoder operates on full codes $\{b_1, b_2, \ldots, b_T\}$

$$\begin{aligned} \hat{I}_0^{(2)} &= 0, & \hat{I}_t^{(2)} &= D\left(b_t, h_{t-1}^{(2)}\right) + \hat{I}_{t-1}^{(2)}, \\ O_{t+T} &= O_{t+T-1} + \hat{I}_t^{(2)} - \hat{I}_t^{(1)}, & &\text{for } t = 1, 2, \ldots T \end{aligned} \tag{4.5}$$

to refine background.

In the first pass, we do not need the original bit values at spacial location (i, j) where $M_{i,j} = 0$. They are always filled with value 0. Therefore, the bitrate at each iteration is lower than the original framework. At the same bitrate, our method provides clearer appearance of the semantic salient objects.

Chapter 5

Experiments

We validate our approach for progressive style encoding and semantic progressive decoding.

5.1 Datasets and Protocol

We train our models on MS COCO 2014 dataset [11], which contains 82k training images. We randomly select 100 images from MS COCO 2014 validation set as a testing set for human evaluation. Furthermore, we test our method on the widely used Kodak dataset [1], a set of 24 uncompressed images. During testing, we resize the image so that both the width and height is a multiple of 16 to fit the network structure. We do this for simplicity. A better solution is to keep the original image solution and zero-pad the input.

Following [10, 13, 21], we evaluate our methods based on the compression rate in bits per pixel (BPP), and image quality in multi-scale structural similarity (MS-SSIM) [23] and peak signal-to-noise ratio (PSNR). MS-SSIM correlates better with human perception of distortion than PSNR. However, it is only defined on grayscale images. We can still compute MS-SSIM on RGB images, but it will be more sensitive to structures than colors compared to PSNR.

We progressively encode all images to BPP 1.25. We report the average performance over the images during progressive decoding. We do not take file header into account when calculating BPP in all our comparison methods.

5.2 Training Details

We train our models for 20 epochs using ADAM [9] optimizer. We use a batch size of 8 and a learning rate 0.0005, which is divided by 2 at epoch 3 and 10. We train on 128×128 random crops in RGB color space, and normalize input values to range $[-0.5, 0.5]$. No other data augmentation is used. All the models are trained with $T = 10$ reconstruction iterations. Note that we do not use mask during training.

5.3 Progressive Style Encoding

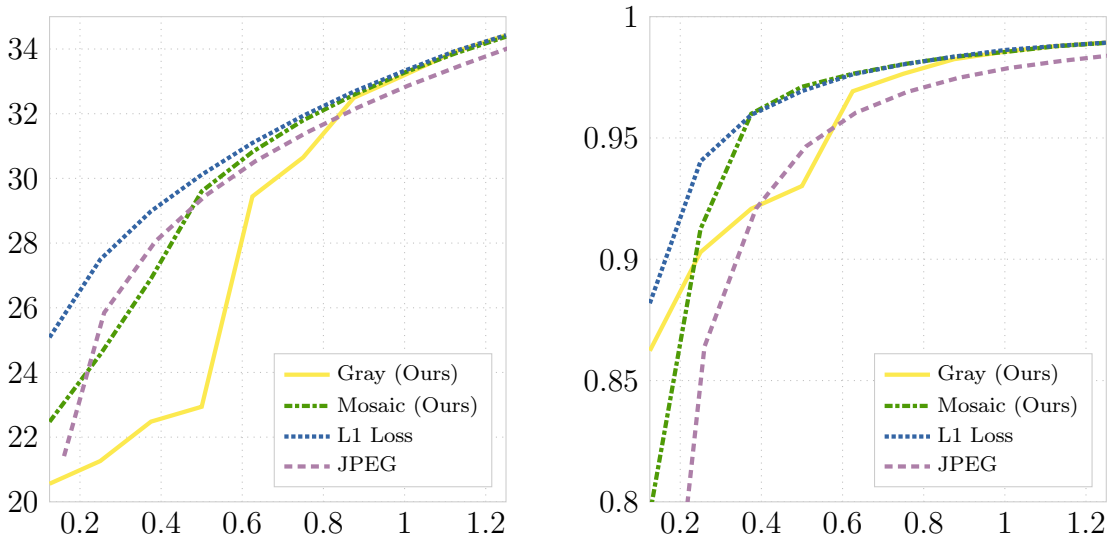
We compare our methods to the following baselines:

- JPEG: We use libjpeg [2] in the codec's default 4:2:0 chroma subsampling. A precise comparison is using progressive JPEG to compress an image to the final target bitrate and snapshotting each reconstructed image during progressive decoding. However, it requires some efforts to hack into the codec to get intermediate images during decoding. For simplicity, we use normal JPEG to compress an image at multiple bitrates, approximating the sequence of reconstructed images during progressive decoding.

- L1 Loss: The model from Toderici *et al.* [21] that optimizes L1 loss between reconstructed images and original images.

5.3.1 Results

We evaluate grayscale to color encoding (Gray) and mosaic style encoding (Mosaic) against the baselines on the Kodak dataset. Figure 5.1 shows the results. Both of our methods achieve the same visual quality as the L1 Loss model on final



(a) PSNR (dB)

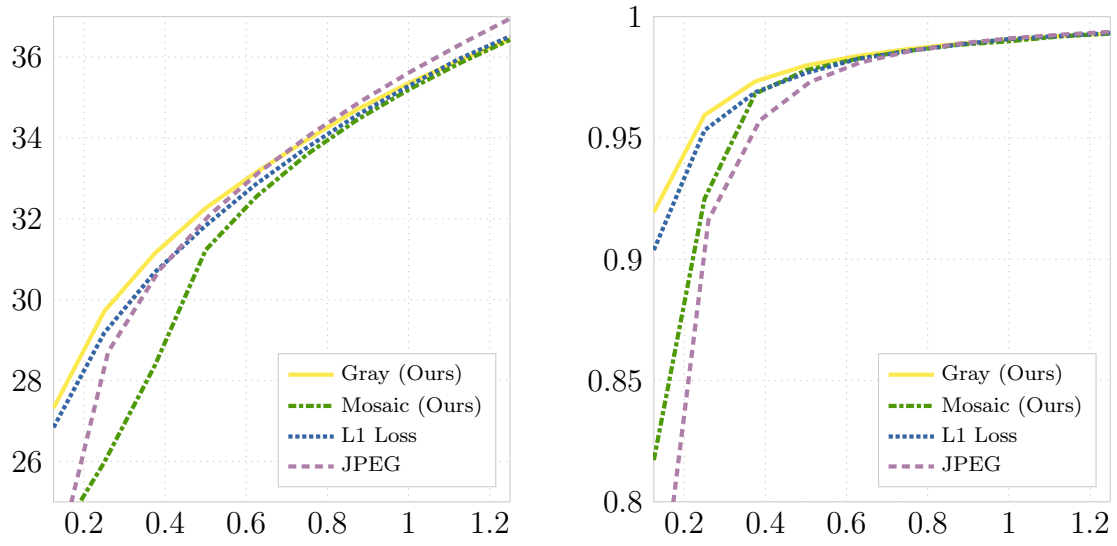
(b) MS-SSIM

Figure 5.1: Rate-distortion curves on the Kodak dataset. Deep image compression methods achieve higher visual quality on final reconstruction.

reconstruction. This validates that our encoding methods do not affect the quality of final decompressed image, but only change the visual progress during decoding. Our goal is only to have a better reconstruction at the low bitrate regime so users can have a clearer preview when only part of the file has been received. We do not aim to achieve higher quality on fully loaded image.

At the low bitrate regime and in terms of PSNR, grayscale to color encoding has the worst performance, and mosaic encoding only performs on par with JPEG. This is because PSNR assumes pixel-wise independence and penalizes the loss of color heavily. However, our methods outperform JPEG nearly across all bitrates in MS-SSIM, which emphasizes structures of images.

In addition, deep image compression methods (Gray, Mosaic, and L1 Loss) outperform JPEG, one of the most common image format on the web, on final reconstruction. This suggests that it is promising to extend deep learning based compres-



(a) PSNR (dB)

(b) MS-SSIM

Figure 5.2: Rate-distortion curves for the luma component only on the Kodak dataset. Grayscale to color encoding displays the sharpest and the clearest images in early decoding (BPP < 0.6).

sion methods to be more suitable for web use.

Next, we evaluate the sharpness of reconstructed images. A sharper image preview provides more details to let users easily understand the content of the image, so they can decide whether to wait for it to fully loaded. To disregard color, we convert images to Y'CbCr color space, and measure the distortion only on the luma component.

Figure 5.2 shows the results. When only 50% of the image file is decoded, grayscale to color encoding displays the sharpest reconstruction among all methods both in PSNR and in MS-SSIM. On the other hand, mosaic style encoding does not improve sharpness at all, because mosaic style mainly blurs images.

Finally, we present qualitative results of all progressive coding methods in Figure 5.3 and Figure 5.4. Grayscale to color model presents more details at the low bitrate regime than the L1 Loss model. For example, human faces, human bodies,

and trees. It smoothly shows color emerging effects. In the intermediate steps, an image is a mix of color and black-and-white information. Mosaic style model generates reconstructions from low resolution to high resolution. It is blurry at very low bitrates. JPEG shows serious color degradation at 0.125 BPP while other methods do not suffer from this problem.

5.3.2 User Studies

To test how clear the reconstructed images make a viewer to understand the content of images, we conduct pairwise comparisons between JPEG, L1 Loss, grayscale to color encoding, and mosaic style encoding. We compare images that are randomly chosen from MS COCO 2014 validation set at the low bitrate, 0.125 BPP. The participants are given a pair of image at a time, and are asked which of the two contains clearer or sharper details. We have 4 participants. Each viewed 100 pairs in each pairwise comparison, and totally viewed 600 pairs in the six pairwise comparisons. It took about 40 minutes for a participant to finish the study.

Figure 5.5 shows the results. We report the percentage of times each method is chosen as the preferred one. We can see that the user studies are consistent with our quantitative analysis. At the low bitrate regime, grayscale to color encoding produces clearer details of the content than L1 Loss model, JPEG, and mosaic style encoding.

5.4 Semantic Progressive Decoding

We compare the effects of masking in decoding to three baselines: L1 Loss model, grayscale to color model, and mosaic style model. We use the segmentation masks from MS COCO dataset, and only use the masks containing person. We use

max pooling to downsample the mask to the spatial resolution of the bottleneck. To make sure that the downsampled mask will fully cover the region we want in the orginal resolution, we dilate the mask by a 32×32 kernel before downsampling it.

5.4.1 Results

Figure 5.6 shows the visual examples of our approach. Our semantic progressive decoding (middle column) clearly produces better reconstruction of human face, which is the semantic salient region specified by the mask. Figure 5.6a shows the results of L1 Loss model. Without masking, original decompressed images are all blurry. With masking in decoding, our images creates the effect like shallow depth of field. We put background out of focus but make faces in focus. Figure 5.6b shows the results of grayscale to color model. It creates an interesting effect of color mixing, where background is black-and-white but the focused objects are of color. Besides, the network automatically produces blending between grayscale regions and color regions, so the resulting images look natural. Figure 5.6c shows the results of mosaic style model. One thing different from the above models is that masking does not blur the background. We only feed the code for background in the first iteration. In the following iterations, the network automatically makes background less pixelated even when no code for background is fed. We hypothesize that it is a result caused by the state of the recurrent component in our decoder. Our mosaic style decoder learns to generate different resolutions at different iterations, regardless of input codes. In conclusion, our semantic progressive decoding helps emphasize important regions, and also creates interesting artistic effects. Users can specify their own masks to create customized decoded images.

5.4.2 User Studies

To test whether humans are more sensitive in the distortion of important objects, we conduct user studies to compare the results of normal progressive decoding and the results of semantic progressive decoding. We use the same 100 images from MS COCO 2014 validation set as the first user study, and decode the images to a low bitrate, 0.25 BPP. The participants are given a pair of image at a time, and are asked which of the two is perceptually preferred. We have 4 participants. Each viewed 100 pairs in each comparison, and totally viewed 300 pairs in the three comparisons. It took about 25 minutes for a participant to finish the study.

Figure 5.7 shows the results. We report the percentage of times each method is chosen as the preferred one. Our approach outperforms the baseline method in all three comparisons at least 58% of the time. In the study for L1 Loss model, 3 participants chose our approach around 61%-68% of the time, while only 1 participant chose ours 41% of the time. We were interested in this discrepancy so we asked for feedback from participants. Those who chose our method at higher rates said their criteria was whether the important region was clear. In our setting, it is person. The one who prefers our method at a lower rate said the text in images and objects other than humans sometimes were important. Although it is a subjective test, still most viewers care about semantic salient regions and can ignore a little degradation in other regions. In other two studies, all participants prefer our method at a high percentage of times. In the grayscale to color model, one possible reason is that the color on important objects reinforces our sensitivity to them and distracts us from background. In the mosaic style model, the background does not degrade due to the state of recurrent network in decoder, so our approach is selected most of the time.

(a) JPEG

(b) L1 Loss

(c) Grayscale to Color

(d) Mosaic Style

Figure 5.3: Comparison of progressive encoding results at 0.125 BPP, 0.25 BPP. Results of grayscale to color model contain more details of face, objects, and background than those of the L1 Loss model. They also do not contain blocky, ringing artifacts as JPEG. (Best viewed on screen.)

(a) JPEG

(b) L1 Loss

(c) Grayscale to Color

(d) Mosaic Style

Figure 5.4: Comparison of progressive encoding results at 0.5 BPP and 0.75 BPP. View with figure 5.3 to see the sequentially changing results. (Best viewed on screen.)

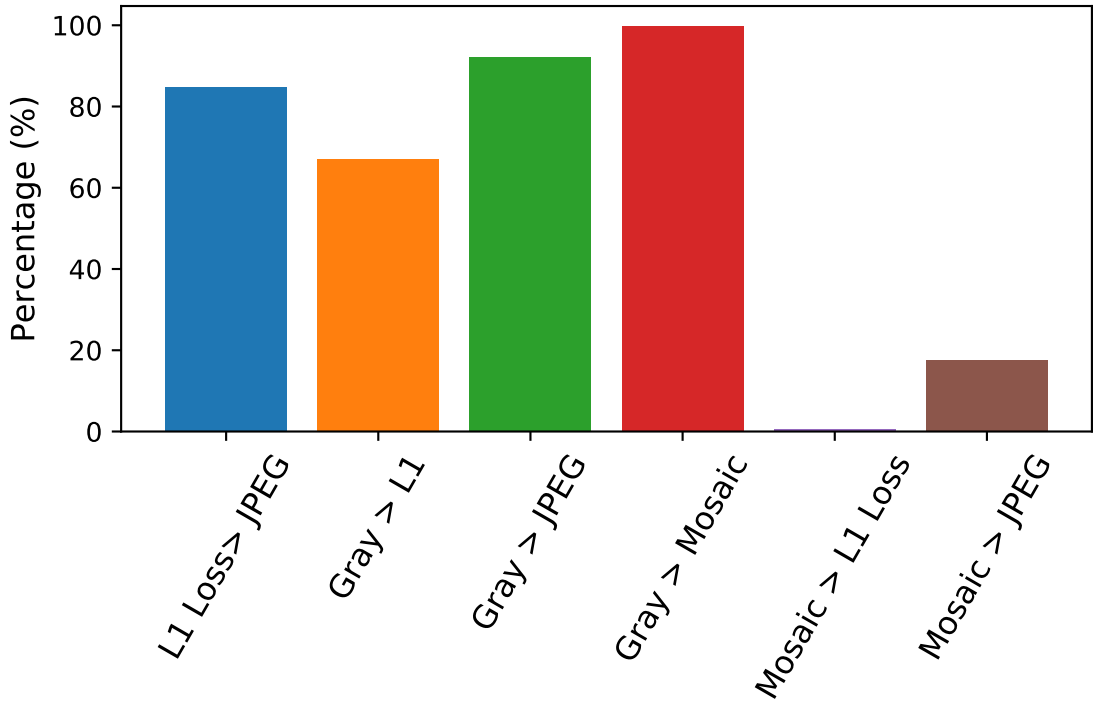


Figure 5.5: User studies to test how viewers perceive decoded images at the low bitrate regime.

(a) L1 Loss.

(b) Grayscale to Color.

(c) Mosaic Style.

Figure 5.6: Semantic progressive decoding results at 0.25 ± 0.01 BPP. We show original reconstruction without masking (left), reconstruction with masking (middle), and the corresponding masks (right).

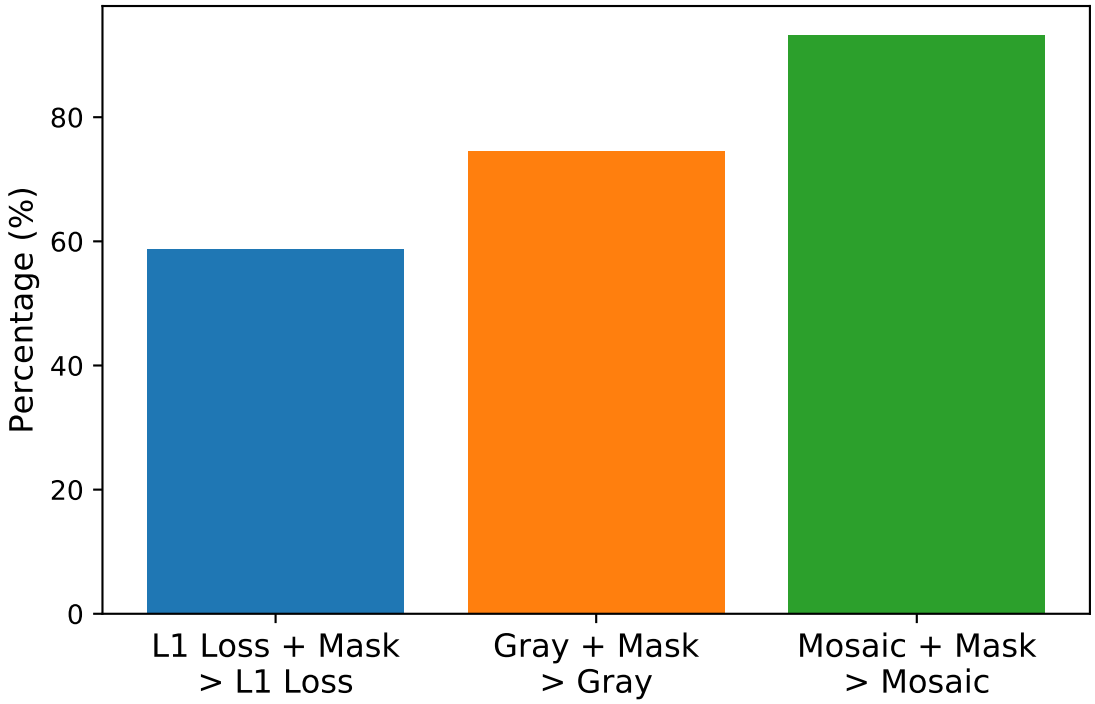


Figure 5.7: User studies to test how viewers perceive semantic progressively decoded images at the low bitrate regime.

Chapter 6

Conclusion

We present an approach to improve progressive image coding at the low bitrate regime. It relies on creating different artistic visual effects and highlighting semantic salient regions. Our approach offers a clearer preview of an image when only a small percentage of file is transmitted. Moreover, users can specify their own spatial decoding patterns. The approach outperforms prevailing traditional compression method, JPEG, and previous state-of-the-art learning based progressive compression method.

Appendices

Appendix A Model Architecture

Table A.1, Table A.2, and Table A.3 show the architecture of our model. The kernels for convolving hidden states in LSTMs have size 1×1, except for the last two Conv-LSTM layers in the decoder where the kernels for hidden states are 3×3. No nonlinearity is used after the first convolutional layers in the encoder and decoder. For the decoder, the last convolutional layer is followed by a tanh nonlinearity to predict RGB values. We divide values after tanh by 2 to make values fall in the range $[-0.5, 0.5]$, which is the same value range for inputs. For the binarizer, the convolutional layer is followed by a tanh nonlinearity, and the binarization $b(x)$ to predict values in $\{-1, 1\}$. We do not include bias in all of the components. We experimented with bias in the model but did not find any difference.

Layer	Activation size
Input (floats)	128×128×3
3×3×64 Conv, pad 1, stride 2	64×64×64
3×3×256 Conv-LSTM, pad 1, stride 2	32×32×256
3×3×512 Conv-LSTM, pad 1, stride 2	16×16×512
3×3×512 Conv-LSTM, pad 1, stride 2	8×8×512

Table A.1: Network architecture of the encoder.

Layer	Activation size
Encoded (floats)	8×8×512
1×1×32 Conv, pad 0, stride 1	8×8×32

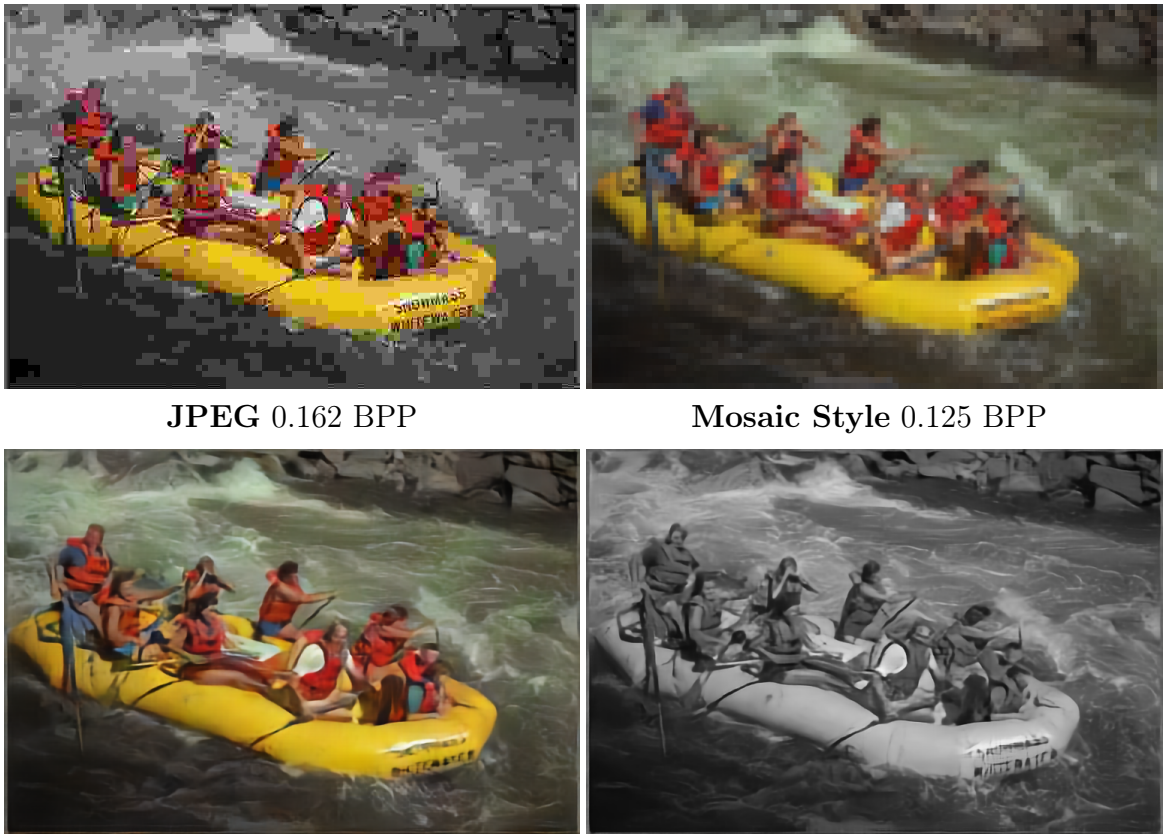
Table A.2: Network architecture of the binarizer.

Layer	Activation size
Binary code (bits)	8×8×32
1×1×512 Conv, pad 0, stride 1	8×8×512
3×3×512 Conv-LSTM, pad 1, stride 1	8×8×512
Depth to Space, stride 2	16×16×128
3×3×512 Conv-LSTM, pad 1, stride 1	16×16×512
Depth to Space, stride 2	32×32×128
3×3×512 Conv-LSTM, pad 1, stride 1	32×32×256
Depth to Space, stride 2	64×64×64
3×3×128 Conv-LSTM, pad 1, stride 1	64×64×128
Depth to Space, stride 2	128×128×32
1×1×3 Conv, pad 0, stride 1	128×128×3

Table A.3: Network architecture of the decoder.

Appendix B Visual Examples

We show additional qualitative results of our approaches. Figure B.1, Figure B.2, and Figure B.3 show example images from the Kodak dataset compressed by different approaches: JPEG, L1 Loss model, Grayscale to Color model, and Mosaic Style model. Figure B.4, Figure B.5, and Figure B.6 show example images from MS COCO 2014 validation set compressed by the same approaches. Figure B.7, Figure B.8, and Figure B.9 show the example results of semantic progressive decoding from L1 Loss model, Grayscale to Color model, and Mosaic Style model respectively.



JPEG 0.162 BPP

Mosaic Style 0.125 BPP

L1 Loss 0.125 BPP

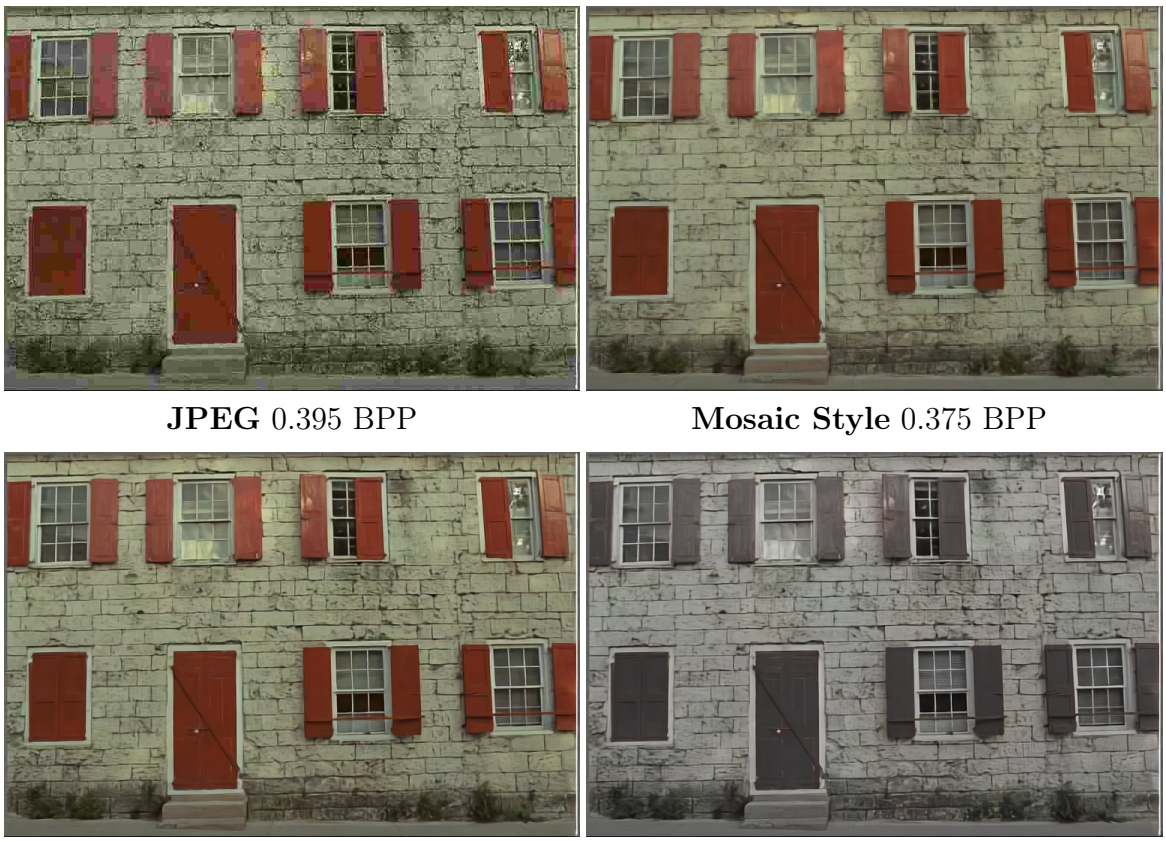
Grayscale to Color 0.125 BPP

Figure B.1: Comparison of progressive encoding results on Kodak Image 14.

JPEG 0.254 BPP **Mosaic Style** 0.25 BPP

L1 Loss 0.25 BPP **Grayscale to Color** 0.25 BPP

Figure B.2: Comparison of progressive encoding results on Kodak Image 9.



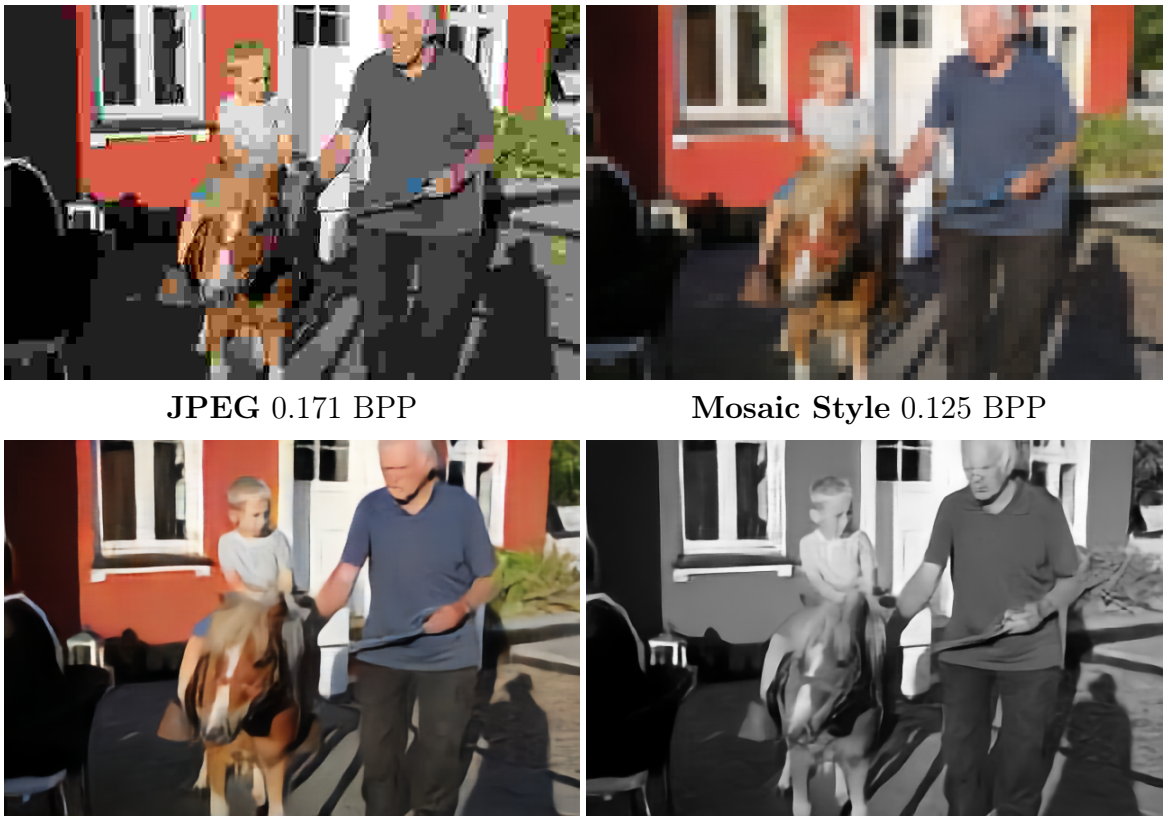
JPEG 0.395 BPP

Mosaic Style 0.375 BPP

L1 Loss 0.375 BPP

Grayscale to Color 0.375 BPP

Figure B.3: Comparison of progressive encoding results on Kodak Image 1.



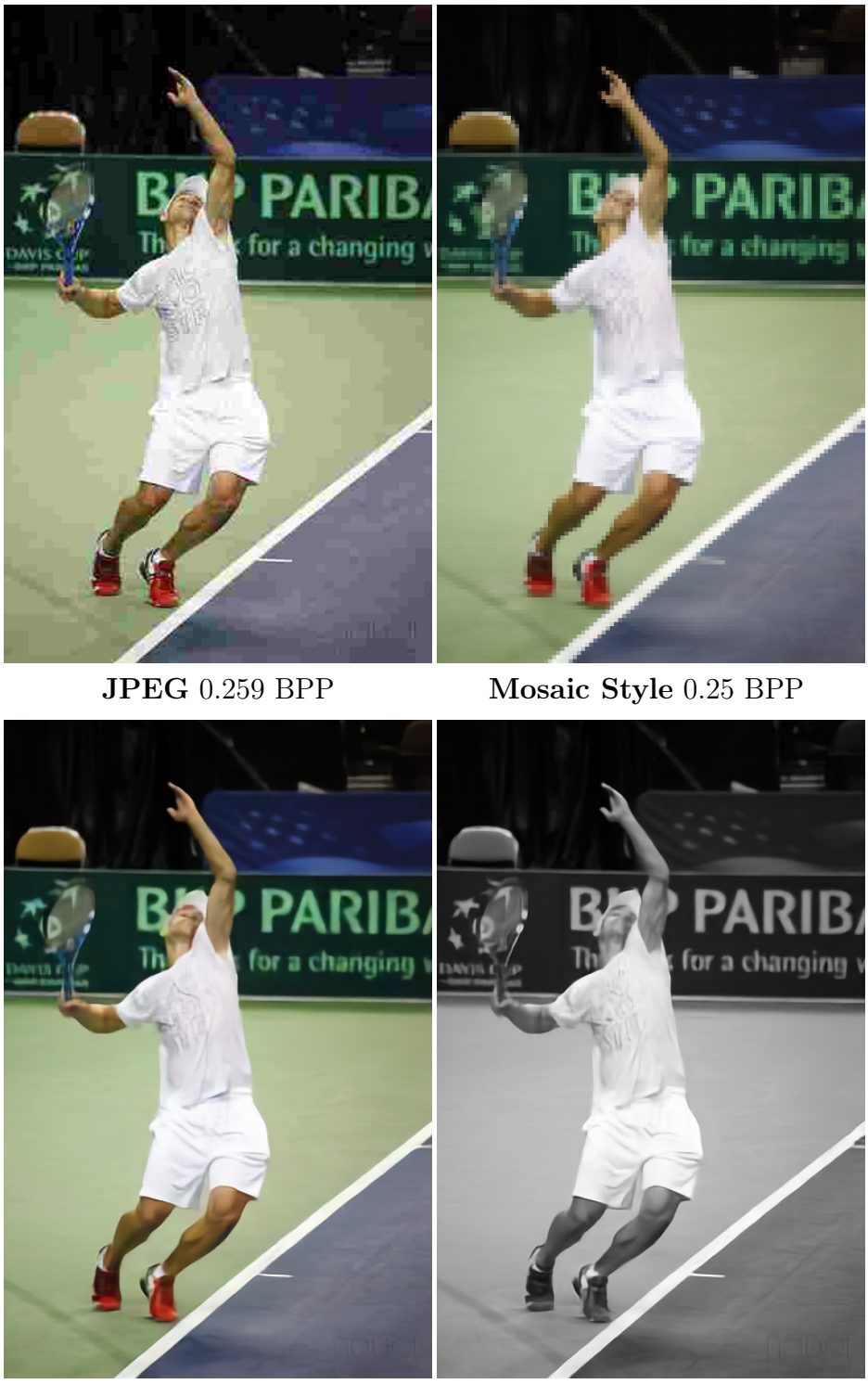
JPEG 0.171 BPP

Mosaic Style 0.125 BPP

L1 Loss 0.125 BPP

Grayscale to Color 0.125 BPP

Figure B.4: Comparison of progressive encoding results on the image from MS COCO validation set.



JPEG 0.259 BPP **Mosaic Style** 0.25 BPP

L1 Loss 0.25 BPP **Grayscale to Color** 0.25 BPP

Figure B.5: Comparison of progressive encoding results on the image from MS COCO validation set.

JPEG 0.381 BPP | **Mosaic Style** 0.375 BPP

L1 Loss 0.375 BPP | **Grayscale to Color** 0.375 BPP

Figure B.6: Comparison of progressive encoding results on the image from MS COCO validation set.

0.375 BPP 0.359 BPP

0.5 BPP 0.451 BPP

0.5 BPP 0.491 BPP

Figure B.7: Semantic progressive decoding results of L1 Loss model. We show original reconstruction without masking (left), reconstruction with masking (middle), and the corresponding masks (right).

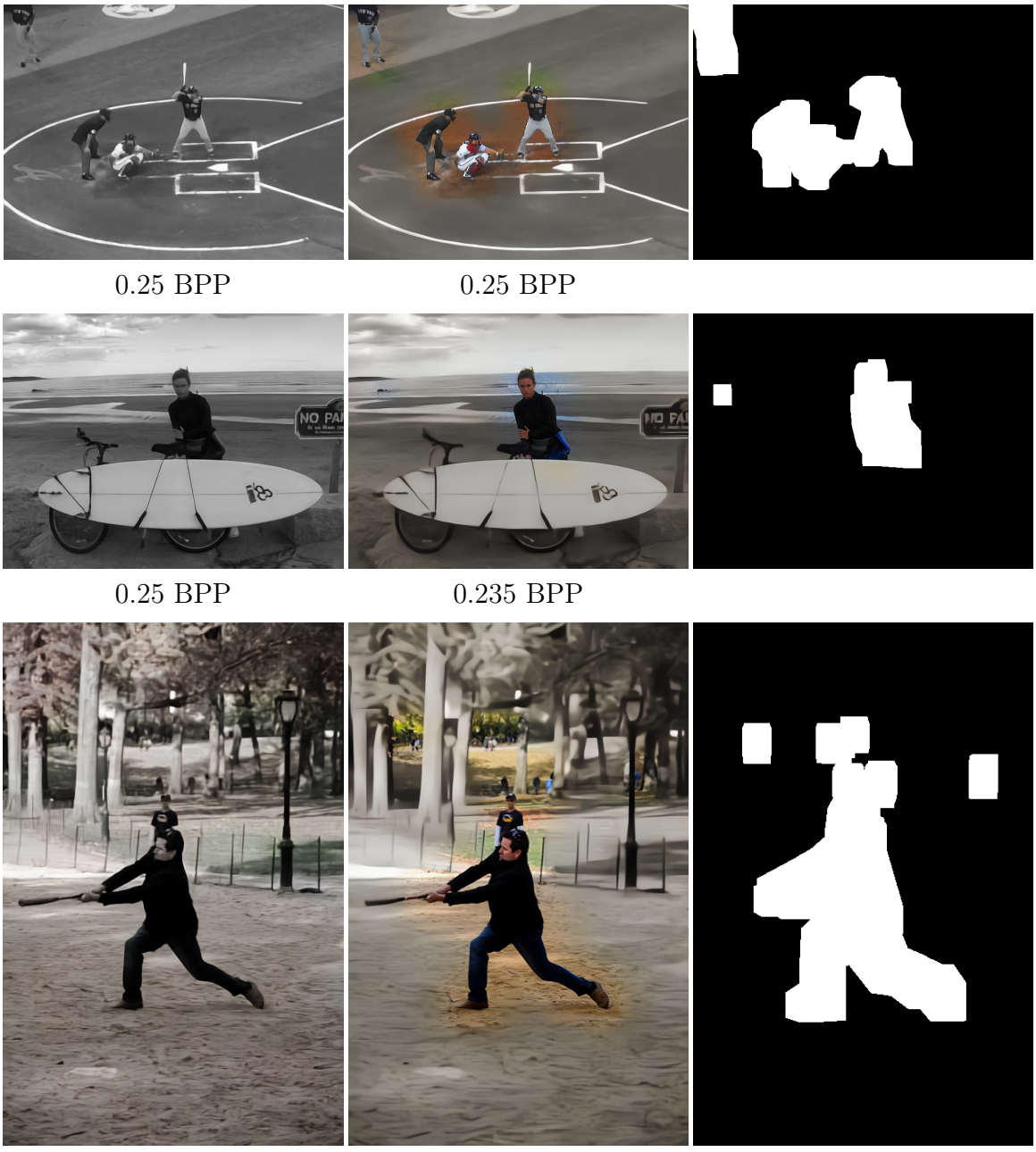


Figure B.8: Semantic progressive decoding results of Grayscale to Color model. We show original reconstruction without masking (left), reconstruction with masking (middle), and the corresponding masks (right).

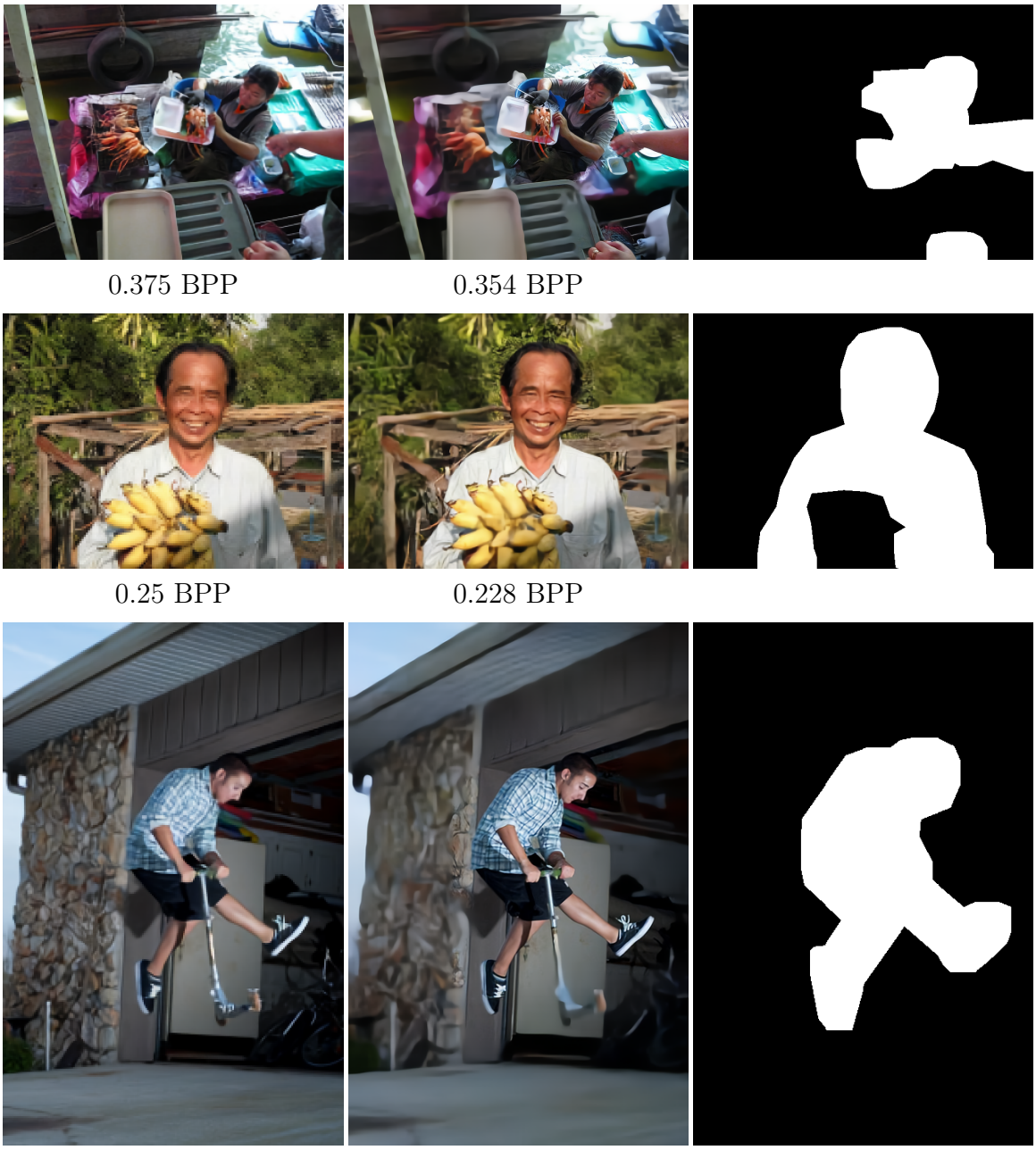


Figure B.9: Semantic progressive decoding results of Mosaic Style model. We show original reconstruction without masking (left), reconstruction with masking (middle), and the corresponding masks (right).

Bibliography

[1] Kodak PhotoCD dataset. http://r0k.us/graphics/kodak/. Accessed: 2019-04-23.

[2] libjpeg. http://libjpeg.sourceforge.net/. Accessed: 2019-04-23.

[3] Web Page Performance Test for Twitter. https://www.webpagetest.org/performance_optimization.php?test=170717_NQ_1K9P&run=2#compress_images. Accessed: 2019-04-24.

[4] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. Van Gool. Generative adversarial networks for extreme learned image compression. *arXiv preprint arXiv:1804.02958*, 2018.

[5] S. Arthur. Making Photos Smaller Without Quality Loss. https://engineeringblog.yelp.com/2017/06/making-photos-smaller.html. Accessed: 2019-04-24.

[6] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *ICLR*, 2016.

[7] T. Bar. Faster Photos in Facebook for iOS. https://code.fb.com/ios/faster-photos-in-facebook-for-ios/. Accessed: 2019-04-24.

[8] F. Bellard. BPG Image Format. https://bellard.org/bpg/. Accessed: 2019-04-13.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang. Learning convolutional networks for content-weighted image compression. In *CVPR*, 2018.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár,

and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[12] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):620–636, 2003.

[13] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Conditional probability models for deep image compression. In *CVPR*, 2018.

[14] J. Murphy and M. Roser. Internet. *Our World in Data*, 2019. https://ourworldindata.org/internet.

[15] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[16] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. Storer. Semantic perceptual image compression using deep convolution networks. In *DCC*, 2017.

[17] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *ICML*, 2017.

[18] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, 2016.

[19] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001.

[20] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. In *ICLR*, 2015.

[21] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. In *CVPR*, 2017.

[22] G. K. Wallace. The jpeg still picture compression standard. *IEEE transactions*

on consumer electronics, 38(1):xviii–xxxiv, 1992.

[23] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *ACSSC*, 2003.

[24] S. Winkler, M. Kunt, and C. J. van den Branden Lambrecht. Vision and video: models and applications. In *Vision Models and Applications to Image and Video Processing*, pages 201–229. Springer, 2001.